Hierarchical Reinforcement Learning under Mixed Observability

Hai Nguyen[1*], Zhihan Yang[2*], Andrea Baisero[1], Xiao Ma[3], Robert Platt[1†], and Christopher Amato[1†]
∗ Equal contribution † Equal Advising

[1] Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA
[2] Calerton College, Northfield, MN, USA
[3] National University of Singapore, Singapore
nguyen.hai1@northeastern.edu

Abstract. The framework of mixed observable Markov decision processes (MOMDP) models many robotic domains in which some state variables are fully observable while others are not. In this work, we identify a significant subclass of MOMDPs defined by how actions influence the fully observable components of the state and how those, in turn, influence the partially observable components and the rewards. This unique property allows for a two-level hierarchical approach we call HIerarchical Reinforcement Learning under Mixed Observability (HILMO), which restricts partial observability to the top level while the bottom level remains fully observable, enabling higher learning efficiency. The top level produces desired goals to be reached by the bottom level until the task is solved. We further develop theoretical guarantees to show that our approach can achieve optimal and quasi-optimal behavior under mild assumptions. Empirical results on long-horizon continuous control tasks demonstrate the efficacy and efficiency of our approach in terms of improved success rate, sample efficiency, and wall-clock training time. We also deploy policies learned in simulation on a real robot.

Keywords: Robot Learning, Hierarchical, Mixed Observability

1 Introduction

Many robotic domains feature a state space that factorizes into high and low observability subspaces, in which actions primarily influence the high observability components of the state. For example, robot navigation with unknown dynamics and noisy sensors to an unknown dynamic target [5] (Fig. 1a), or robot manipulation to reach an unknown target pose, *e.g.*, find an object in cluttered and occluded environments [29] (Fig. 1b), grasp under uncertainty [13], or collaborate with humans with partially observed human factors (*e.g.*, trust [3], preferences [27], or goals [21]). In these examples, state variables such as a robot's position or an arm's pose can be measured with high accuracy (*e.g.*, using GPS signals and sensors) than partially observable variables relative to the task and

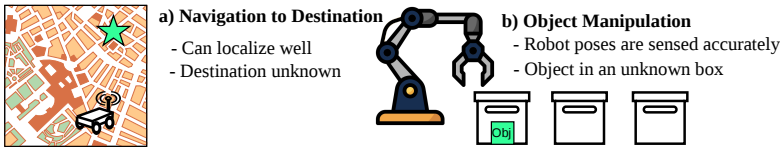


Fig. 1: Examples of partially observable domains of our interests.

are often assumed fully observable. Moreover, actions directly influence the fully observable state components and indirectly affect partially observable ones. For example, in Fig. 1, navigation actions can lead the mobile robot to different locations that contain useful information to reach the destination; sequences of poses help the robot arm open boxes to examine which contains the object.

Our contributions are as follows. First, we formulate such tasks using the framework of mixed observability Markov decision process (MOMDP) [22] and define motion-based MOMDPs (MOB-MOMDPs), a subclass of MOMDPs which makes mild assumptions concerning dynamics and tasks. Second, we introduce a hierarchical solution to solve MOB-MOMDPs, consisting of a high-level policy with partial observability and a low-level policy with full observability. In our agent, the top policy uses high-level observations to compute the bottom policy's goals, which are desired points in the fully observable space. When the bottom policy achieves a given goal or times out, emitted observations are used to produce the next high-level observation for selecting the next goal, and so on. Our approach can potentially offer efficient learning by breaking a long-horizon task into easier-to-learn subtasks, which enjoy full observability. Moreover, a hierarchical approach would explore more efficiently when rewards are sparse, thanks to high-level actions. In both theory and empirical experimentation, we show that our proposed hierarchical approach can achieve optimal or quasi-optimal behavior in MOB-MOMDPs with sufficiently low stochasticity constraints.

We demonstrate the benefits of our approach on long-horizon simulated continuous control domains with sparse rewards. Such domains are challenging for many non-hierarchical POMDP methods [9, 19]. In contrast, our hierarchical agent achieves higher success rates with excellent efficiency in training samples and wall-clock training time. Further, our robot experiments show that a learned policy could be effectively deployed in the real world.

2 Background

In this section, we will first go through the background of a goal-conditioned MDP (which will be solved by the bottom level), then the frameworks of partially observable Markov decision processes (POMDP) and MOMDP. We conclude by the hierarchical reinforcement learning algorithm that our approach builds upon.

A **goal-conditioned MDP** is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{G}, T, R, \gamma)$, where S is the state space, $\mathcal{A}$ is the action space, $\mathcal{G}$ is the goal space, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the

transition function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. The objective is to find a goal-conditioned policy $\pi \colon \mathcal{S} \times \mathcal{G} \to \Delta\mathcal{A}$ which maximizes the return $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$, where r_t is the reward at timestep t.

A **POMDP** [2] is specified by a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma)$, where ($\mathcal{S}$, $\mathcal{A}$, T, R, γ) are the same as in a goal-conditioned MDP. Instead of directly observing the state s, the agent only observes $o \in \Omega$ after taking an action a and reaching state s' governed by the observation function $O(s', a, o) = p(o \mid s', a)$. The goal is to find a policy π that maximizes the expected discounted return defined as $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$. To take an optimal action at timestep t, an agent often must condition its policy on the entire action-observation history $h_t = (o_{\leq t}, a_{<t}) \in \mathcal{H}_t$ that it has seen so far. However, the size of the history $\mathcal{H}_t$ grows exponentially with t. Therefore, a recurrent neural network (RNN) is often used to summarize h_t with its fixed-sized hidden state.

A **MOMDP** [22] is a POMDP in which a state can be decomposed as $s = (x, y)$, where $x \in \mathcal{X}$ is fully observable and $y \in \mathcal{Y}$ is partially observable. Since x is fully observable, an observation o can be decomposed as $o = (x, z) \in \Omega$ where z is the remaining component of o. The observation function

$$O(s', a, o) = p(o \mid s', a) = p(x, z \mid x', y', a) = \mathbb{1}[x = x']p(z \mid x', y', a) \tag{1}$$

specifies which observation the agent gets after it took action a and reached state $s' = (x', y')$ with $\mathbb{1}$ denoting the indicator function. The transition function $T(s, a, s') = p(s' \mid s, a) = p(x', y' \mid x, y, a)$ for $x, x' \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$ specifies the probabilities of reaching a state (x', y') after taking an action a in state (x, y). $T(s, a, s')$ can be decomposed as

$$T(s, a, s') = p(s' \mid s, a) = p(x', y' \mid x, y, a) = p(x' \mid x, y, a)p(y' \mid x, y, a, x'). \tag{2}$$

Let $T^{\mathcal{X}}(x, y, a, x') = p(x' \mid x, y, a)$ and $T^{\mathcal{Y}}(x, y, a, x', y') = p(y' \mid x, y, a, x')$, the tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{A}, T^{\mathcal{X}}, T^{\mathcal{Y}}, R, \Omega, O, \gamma)$ formally defines a MOMDP.

Hierarchical Actor-Critic (HAC) [16] is an MDP hierarchical agent, in which an action from a non-base level is a goal for the policy at the level right below it, and the base policy will directly interact with the environment. The policies in each level are trained in an off-policy manner using replay buffers, one for each level. We build our agent upon a two-level HAC agent and utilize different techniques in HAC to stabilize the training at the top level and learn effectively under sparse rewards at the bottom. More details are in Section 4.

3 Motion-Based MOMDPs

We define motion-based MOMDPs (MOB-MOMDPs) as MOMDPs which satisfy the following additional factorization and independence assumptions,

$$p(s' \mid s, a) = p(x' \mid x, a)p(y' \mid x, y, x')\,, \tag{3}$$

$$p(o \mid s', a) = p(o \mid s')\,, \tag{4}$$

$$R(s, a) = R(s)\,. \tag{5}$$

In other words, a) the fully observable component x of the state satisfies the Markov property without depending on the partially observable component y of the state, b) both the partially observable component y of the state and the observed component z are conditionally independent on the action a (when conditioned on the fully observable component x of the state), and c) the task is encoded by a reward function which exclusively depends on the reached states, and not the actions taken. We refer to MOB-MOMDPs, which have deterministic (stochastic) $T^{\mathcal{X}}$ as *deterministic* (*stochastic*) MOB-MOMDPs; note that this does not refer to the stochasticity of $T^{\mathcal{Y}}$, which remains unconstrained.

In MOB-MOMDPs, actions only have a direct influence on the resulting x trajectories, while their influence on the y, z, and reward trajectories is indirect through x. MOB-MOMDPs include (but are not limited to) navigation tasks where x represents the fully observable pose of the agent in the environment, while y represents other partially observable information about the environment and task. In such navigation MOB-MOMDPs, actions relate to the *motion* of the agent, and it is exclusively through such motion that the agent is able to interact with the environment, gather information, and complete the task. Although not all MOB-MOMDPs intrinsically represent navigation tasks, we will use the imagery of navigation tasks as a useful analogy to simplify the way we discuss and analyze MOB-MOMDPs and the respective learning algorithms. Therefore, we reinterpret general MOB-MOMDPs as navigation tasks where x figuratively represents the agent's pose, a the movements that allow the agent to change its pose, and y as any other partially observable aspect concerning the task.

Using such analogy, and because actions exclusively influence the environment through the resulting agent pose, it is possible to abstract "motion"-based control (*i.e.*, based on the actions which move the agent) as "pose"-based control (*i.e.*, based on the poses which the agent should reach in order to gain information or complete the task). "Pose"-based control is executed not by choosing how the agent should move (action a), but rather where it should move to (pose x'). Such abstraction is the inspiration for a flavor of hierarchical reinforcement learning specifically suited to solve MOB-MOMDPs.

4 Hierarchical Reinforcement Learning under Mixed Observability

We first give an overview of our hierarchical method, and how each hierarchy layer is trained. Then we provide an optimality analysis of the approach.

4.1 Approach

Overview. As shown in Fig. 2, HIerarchical reinforcement Learning under Mixed Observability (HILMO) makes decisions through a two-level hierarchy. The top-level policy is a recurrent module (symbolized by an arrow pointing to itself) which takes in a top-level observation o_t^T (a summary of several past primitive observations $o \in \Omega$) to produce a goal $x_t^g \in \mathcal{X}$ being the desired value

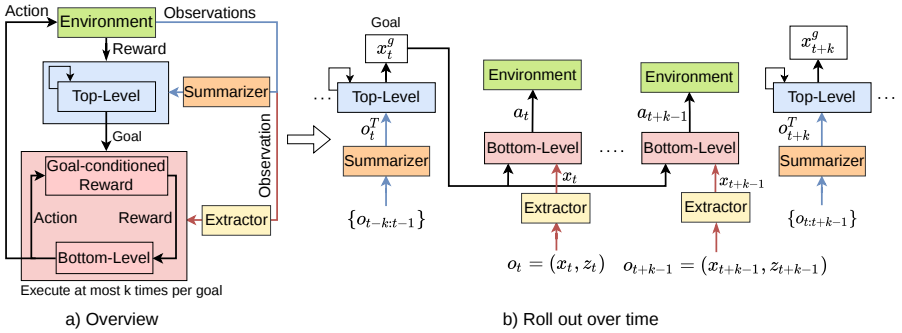


Fig. 2: Our two-level hierarchical agent. A *memory-based* top-level policy looks at a summary of several past observations to select a desired state (goal) x_t^g for the bottom-level policy. A *memoryless* bottom policy then looks at x_t (a component of observation $o_t = (x_t, z_t)$) and the goal state x_t^g to produce at most k primitive actions to achieve x_t^g, emitting a new sequence of observations. Next, these observations will be fed to a summarizer to create a new high-level observation for the top policy to select a new goal.

for x_t. Then the memoryless bottom-level policy selects an action a_t using x_t (extracted from $o_t = (x_t, z_t)$) and x_t^g. Notice this also covers the case when we want to set the goal only for a subspace of $\mathcal{X}$. For instance, although x of the mobile robot in Fig. 1 might include both positions and velocities, we might just want it to successfully reach a certain position regardless of its velocity.

The goal x_t^g remains unchanged until x_t^g is achieved or k bottom-level actions have been performed (for clean notations, from now, we assume that the bottom episode will *always* last k timesteps). When the bottom level finishes acting, k observations $o_{t:t+k-1}$ (*i.e.*, $o_t, \ldots, o_{t+k-1}$) are emitted. These observations will be fed to a summarizer to create the next top-level observation o_{t+k}^T for the top policy to choose the next goal. In this hierarchy, the top level acts at a higher temporal resolution than the bottom level. For a complete algorithm of HILMO, please refer to Appendix A.

Bottom-Level. A bottom-level goal-conditioned MDP $\mathcal{M}^{\mathcal{X}}$ is specified by $(\mathcal{X}, \mathcal{A}, \mathcal{G} = \mathcal{X}, T^{\mathcal{X}}, R^{\mathcal{X}}, \gamma)$. The reward function $R^{\mathcal{X}}$ is defined for each goal $x^g \in \mathcal{X}$ as $R^{\mathcal{X}}(x', x^g) = -\mathbb{1}[d(x', x^g) \geq \epsilon]$, where d is some given distance metric in $\mathcal{X}$ and ϵ is a small reaching threshold. The transition function $T^{\mathcal{X}}(x, a, x') = p(x' \mid x, a)$ specifies the probabilities of reaching x' after taking action a in x. A goal-conditioned policy $\pi^{\mathcal{X}}(a \mid x, x^g)$ that solves $\mathcal{M}^{\mathcal{X}}$ will maximize the discounted cumulative reward $\sum_{t'=t}^{t+k-1} \gamma^{t'-t} R^{\mathcal{X}}(x_{t'}, x^g)$, where x_t is the starting state. The input x of $\pi^{\mathcal{X}}$ is from an extractor that extracts x from $o = (x, z)$.

Top-Level. A top-level POMDP $\mathcal{P}^T$ is specified by $(\mathcal{S}, \mathcal{A}^T = \mathcal{X}, T^T, R^T, \Omega^T, O^T, \gamma)$. In particular, its action space $\mathcal{X}$ is the goal space in $\mathcal{M}^{\mathcal{X}}$, therefore a *top-level* policy $\pi^T(\cdot \mid o^T)$ that solves $\mathcal{P}^T$ will output a desired state x^g to be reached by $\pi^{\mathcal{X}}$. The top-level transition function can be specified as a *multi-time*

model [23] that describes a multi-step policy taking in the goal a^T from state s

$$T^T(s, a^T, s') = \sum_{m=1}^{k} p(s', m \mid s, a^T)\,, \tag{6}$$

where $p(s', m \mid s, a^T)$ is the probability that the bottom policy $\pi^{\mathcal{X}}$ terminates at state s' after exactly m primitive actions when it acts to achieve the goal a^T starting from state s. Unlike $\pi^{\mathcal{X}}$, the objective of π^T is to optimize the discounted cumulative reward $\sum_{t=0;t+=k}^{\infty} \gamma^{t/k} R^T(s_t, a_t^T)$ where each $R^T(s_t, a_t^T)$ is the expected accumulated environment rewards when π^X acts for k timesteps to achieve the goal a_t^T starting at state s_t. Top-level observations are defined as the sequence of actions and observations obtained by the low-level policy until control is given back to the high-level policy, e.g., assuming that at time-step t the high-level policy chooses goal $x^g = a^T$, and that the low-level policy interacts with the environment for k timesteps by choosing actions $(a_t, \ldots, a_{t+k-1})$ and receiving observations $(o_t, \ldots, o_{t+k-1})$, then $o^T = (a_t, o_t, \ldots, a_{t+k-1}, o_{t+k-1})$.

4.2 Bottom-level Policy Learning

Because the bottom policy acts in a fully observable system with sparse rewards, we learn it using transitions sampled from a replay buffer using goal relabeling [1].

Goal Relabeling is a commonly used and powerful technique for learning under sparse rewards. We utilize the technique to replace unmet goals in past transitions with ones met in hindsight, creating positive learning signals as goals are met. Similarly, HAC uses goal relabeling to create hindsight goal transitions (HGT). Specifically, given a bottom-level transition (x, a, x', r, x^g) that did not reach x^g but reached x'^g instead, a modified transition (x, a, x', r', x'^g) where r' is the new reward associated with x'^g will be used for training. In contrast, if x^g is reached when the transition ends, the original transition will be used.

Learning Algorithm. Adopting HAC's choice, we use a version of DDPG [18] without target networks. We also experimented with other learning algorithms such as TD3 [7] and SAC [8], but none outperformed DDPG. Please see Appendix B for more details about our implementation of DDPG.

4.3 Top-Level Policy Learning

The top policy is trained off-policy using samples from a replay buffer of top-level episodes. This differs from HAC's memoryless top level, which can be trained using transitions. Here we introduce methods to create top-level observations and forming training episodes before describing the learning algorithm.

Creating Top-Level Observations. In practice, the summarizer applies an operator $\mathcal{F}$ to k low-level observations to create a high-level observation. In this perspective, a high-level observation can be considered as an implication of temporally extended perception. Here, we consider three options for $\mathcal{F}$.

Full. Concatenating all k previous observations to create a top-level observation. If the bottom policy finishes before k timesteps, we simply pad zeros to have k observations.

Final. Using the final (k-th) observation as a top-level observation. This approach is commonly used in MDP hierarchical agents such as HAC and HIRO [20].

Recurrent. Using a separate and learnable recurrent layer to summarize all k previous observations with the final hidden state being a top-level observation.

Creating Stationary Training Episodes. Given a top-level episode in with any goal unachieved, using it directly to learn the top policy will cause non-stationarity. For instance, given the same (o^T, a^T), the bottom policy can achieve other undesired goals, leading to multiple possibilities of (o'^T, r^T). This makes learning $Q(o^T, a^T)$ at the top level non-stationary.

Creating Stationary Transitions: Stationary transitions are created using hindsight action transitions (HAT) [16], which are modified transitions *as if* the bottom policy already converges and always achieves its goals. Specifically, given a transition $(o^T, a^T_{\text{unmet}}, o'^T, r^T)$ with an unmet action (goal) a^T_{unmet}, the transition $(o^T, a^T_{\text{met-by-bottom}}, o'^T, r^T)$ will instead be used, where $a^T_{\text{met-by-bottom}}$ is the goal reached by the bottom policy.

Penalizing Unachieved Goals: Producing unrealistic goals that are unreachable by the current bottom policy should be discouraged. Therefore, whenever the top policy produces an unachieved goal for the bottom policy, it will be penalized by a negative reward with some probability. Specifically, given a transition $(o^T, a^T_{\text{unmet}}, o'^T, r^T)$, the transition $(o^T, a^T_{\text{unmet}}, o'^T, -H^T)$ will be used, where H^T is the time horizon of the top policy. These transitions are known as subgoal testing transitions (STT) [16].

Forming Stationary Episodes: While HATs or STTs alone are sufficient to train a memoryless HAC agent, we must combine those transitions to form episodes for training our recurrent top policy. Fig. 3 illustrates a method transforming a non-stationary episode into its stationary versions. Here a top-level non-stationary episode τ comprises three transitions ϕ_1, ϕ_2, and ϕ_3 in which a goal is only achieved (by the bottom policy) in transition ϕ_2. We a) replace ϕ_1 with ϕ_1^{HAT} to create a stationary episode $\hat{\tau}_1$, b) use HAT for ϕ_1 and STT for ϕ_3 to create episode $\hat{\tau}_2$, or c) use HAT for ϕ_1 and ϕ_3 to create episode $\hat{\tau}_3$. Finally, three stationary episodes $\hat{\tau}_1, \hat{\tau}_2$, and $\hat{\tau}_3$ are used to train π^T.

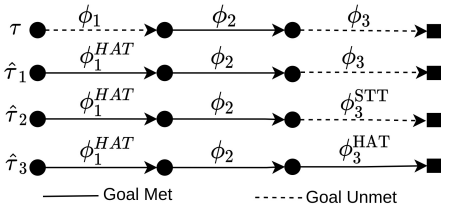


Fig. 3: Creating training episodes for π^T.

Learning Algorithm. We use Recurrent Deterministic Policy Gradient (RDPG) [10] to learn the top-level policy with an LSTM [12] recurrent component. For the algorithm and the implementation details, see Appendix C. We also experiment with Gated Recurrent Unit (GRU) [4] and a recurrent version of TD3 [7] instead of RDPG (see Appendix D for a performance comparison).

4.4 Optimality Analysis

In this section, we analyze the optimality of the HILMO approach in MOB-MOMDPs, *i.e.*, whether the best HILMO policies are also guaranteed to achieve optimal MOB-MOMDP behavior. As it turns out, the amount of stochasticity of the MOB-MOMDP (as defined in Section 3, *i.e.*, only concerning $T^{\mathcal{X}}$) becomes a determining factor on the optimality of the HILMO approach. Initially, we will assume *deterministic* MOB-MOMDPs, and show that optimality is guaranteed.

Theorem 1. *For* deterministic *MOB-MOMDPs and sufficiently large γ, the optimal HILMO policies are optimal or quasi-optimal for the MOB-MOMDP.*

Proof. We first assume $\gamma = 1$, and show by construction that optimal MOB-MOMDP policies can be represented as HILMO policies, and then show that no other HILMO policies are preferable according to the HILMO criteria.

Consider, without loss of generality, a deterministic optimal policy π^* for the *deterministic* MOB-MOMDP. Next, we show that we can construct a HILMO policy which exhibits the same optimal behavior. Given any history h and its associated fully observable pose x, the optimal policy selects action $a = \pi^*(h)$ which causes a transition into pose x'. Due to the deterministic assumptions on $T^{\mathcal{X}}$ and π^*, each h is associated with a unique resulting transition (h, x, a, x'). Consider the HILMO policy $(\pi^T, \pi^{\mathcal{X}})$ constructed such that for each such tuple (h, x, a, x'), $\pi^T(h) = x'$ and $\pi^{\mathcal{X}}(x, x') = a$, while all other actions $\pi^{\mathcal{X}}(x, g)$ can be chosen to achieve the shortest path between x and g to satisfy the low-level MDP optimality criterion. Such HILMO policy exhibits the exact same behavior as the optimal policy π^*, *i.e.*, for all histories h the equality $\pi^{\mathcal{X}}(x, \pi^T(h)) = \pi^*(h)$ holds. This shows that any behavior which is optimal for the control problem can be represented as a HILMO policy. To conclude, we need to show that there is no other HILMO policy that would be preferable to the one constructed according to the HILMO criteria of optimality for the low-level and high-level policies respectively. This is trivial for the constructed low-level policy, which effectively already finds the shortest (i.e., one-step) path between two poses x and x', and for the constructed high-level policy, which shares the same optimality criterion as the original MOB-MOMDP.

When $\gamma < 1$, the HILMO top-level criterion and the MOB-MOMDP criterion apply slightly different forms of discounting. However, for sufficiently large γ, the difference is small enough to ensure that the criteria are either equivalent or approximately equivalent, resulting in optimal or quasi-optimal HILMO policies.

Note that the optimal HILMO policy constructed above executes at the smallest possible temporal scale, *i.e.*, the high-level policy π^T selects goal poses which are directly adjacent to the agent's current pose, and the low-level policy $\pi^{\mathcal{X}}$ is able to reach such goal poses in a single timestep. However, this does not preclude the existence of other optimal HILMO policies which execute at broader temporal scales, in which the high-level policy selects goal poses which require multiple timesteps to be reached.

This analysis is limited to *deterministic* MOB-MOMDPs and does not intrinsically carry over to *stochastic* MOB-MOMDPs. In Appendix E, we extend the analysis, providing a highly stochastic MOB-MOMDP example which demonstrates issues with the HILMO approach and a lowly stochastic MOB-MOMDP, which demonstrates that such issues are minor if the stochasticity is also minor. We argue that because most realistic MOB-MOMDPs have a relatively low amount of stochasticity, the HILMO approach should still be able to achieve theoretically quasi-optimal performance even in mildly stochastic MOB-MOMDPs.

5 Related Work

This section describes prior works on hierarchical reinforcement learning (HRL).

HRL for MDPs. For discrete action spaces, several prior works [6,14] addressed learning hierarchically. For continuous action spaces, HAC [16] and HIRO [20] proposed a hierarchical agent consisting of policies learned jointly in an off-policy manner. HIPPO [17] is an on-policy hierarchical agent but focuses more on optimizing pre-trained skills for downstream tasks.

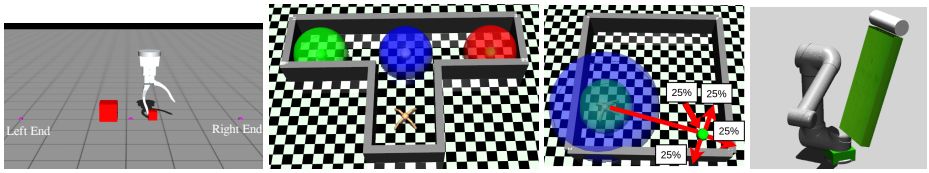
HRL for POMDPs. Hierarchical Suffix Memory [11] incorporated hierarchies with memories to solve navigation tasks under perceptual aliasing. Another two-level hierarchical agent [15] used hand-crafted goals with policies learned independently. HQ-Learning [28] solved specific POMDPs using a pre-specified number of sequentially chained reactive sub-agents. [25] combined hardcoded memoryless options (temporally extended actions) to solve a navigation POMDP by conditioning each option on the previous one. In contrast to these works, which use hardcoded bottom-level policies, hand-crafted goals, and have rigid structures, our hierarchical agent learns policies of all levels jointly, can handle continuous control tasks, and is supported by theoretical analysis.

6 Experiments

We perform experiments on continuous control tasks including two navigation and two manipulation domains implemented in MuJoCo [26]. Below in Table 1, we describe each domain and the corresponding state and observation.

6.1 Domains

Two-Boxes. A finger is velocity-controlled ($\dim(\mathcal{A}) = 1$) on a 1D track to perform a dimension check of two boxes (Fig. 4a). Since the finger is always compliant, it will be deflected from the vertical axis when it glides over a box. The agent observes the finger's position and angle ($\dim(\Omega) = 2$) but not the positions of the two boxes. Therefore, an optimal agent must localize *both* boxes and determine their sizes using the history of angles and positions. When the two boxes have the same size, the agent must go to the right end (pink) to get a non-zero reward and otherwise to the left end. The agent receives a penalty if



a) Two-Boxes b) Ant-Heaven-Hell c) Ant-Tag d) Door-Push

Fig. 4: Four continuous control domains in MuJoCo [26] to perform experiments.

Table 1: State, goal, and observation descriptions. x^g is the goal used by the bottom-level policy in the hierarchical agents (HILMO, HILMO-O, and HAC).

Domain	**Description**
`Two-Boxes`	x, x^g: finger positions, y: desired target position, z: finger angle
`Ant-Heaven-Hell`	x: joint angles and velocities, and body position, x^g: ant body position, y: heaven position, z: heaven position or null
`Ant-Tag`	x: joint angles and velocities, and body position, x^g: ant body position, y: opponent position , z: opponent position or null
`Door-Push`	x: joint angles and velocities, x^g: target joint angles, y: push direction, z: door angle

reaching wrong ends. An episode terminates whenever the two ends are reached, or lasts more than 100 timesteps.

Ant-Heaven-Hell. An ant with four legs ($\dim(\mathcal{A}) = 8$) moving in a 2D T-shaped world will receive a non-zero reward by reaching a green area (heaven) that can be on the left or the right corner (Fig. 4b) of a junction. The ant receives a penalty when entering a red area (hell). When it stays in the blue ball, it can observe heaven's side (left/right/null). Here the observation includes the joints' angles & velocities of the four legs and the side indicator ($\dim(\Omega) = 30$). The ant starts randomly around the bottom corner, and an episode terminates when heaven or hell is reached, or more than 400 timesteps have passed. An optimal agent must visit the blue region to observe heaven's side, memorize the side while going to heaven, and finally goes to heaven.

Ant-Tag. The same ant now has to search and "tag" a moving opponent by being sufficiently close to it (having the opponent inside the green area centered at the ant in Fig. 4c) to get a non-zero reward. Both start randomly but not too close to each other. The opponent follows a fixed stochastic policy, moving a constant distance away from the ant 75% of the time or staying otherwise. An observation includes the joints' angles & velocities of four legs and the 2D coordinate of the opponent ($\dim(\Omega) = 31$), containing the opponent's position only when it is inside the visibility (blue) area centered at the ant. An episode terminates when the opponent is tagged, or more than 400 timesteps have passed.

Door-Push. A 3-DoF gripper ($\dim(\mathcal{A}) = 3$) in 3D must successfully push a door to receive a non-zero reward (Fig. 4d). The door, however, can only be pushed in one direction (front-to-back or vice versa), and the correct push direction is unknown to the agent. Here the agent can observe the joints' angles and velocities and the door's angle ($\dim(\Omega) = 7$). Starting each episode, the door is present to the gripper, initialized with a random pose. An optimal agent must experiment to determine the correct push direction. For example, when it fails to open the door in one direction, it must go to the other side of the door while memorizing the previous push direction that did not work, not to try that again. Therefore, an optimal agent must infer the correct push direction from the history of observations. An episode terminates when the door's angle is larger than a threshold or more than 400 timesteps have passed.

6.2 Agents

We consider the following hierarchical **(H)** and flat **(F)** agents:

- **(H)** A two-level **HILMO** (ours) agent with goals x^g described in Table 1.
- **(H)** A two-level **HAC** [16] agent with the same goal as in HILMO to show that recurrence is needed for a hierarchical agent to solve our domains.
- **(H) HILMO-O** (ours) same structure and goals as HILMO, but implemented using the HIRO [20] framework with off-policy corrections (see Appendix F for the algorithm) to replace HATs. Another difference is that the bottom policy of HIRO originally receives dense instead of sparse rewards, *i.e.*, $R^{\mathcal{X}}(x', x^g) = -d(x', x^g)$, hence HGTs are not used. Plus, there is no penalty for unachieved goals by the top policy (STTs are not used either). Like HAC, HIRO is a common hierarchical baseline for continuous control.
- **(F)** Soft Actor-Critic (**SAC**) [8] with observations instead of states to show that even a strong flat agents cannot solve our domains without memory.
- **(F)** Recurrent Soft Actor-Critic (**RSAC**) [30] is a recurrent version of SAC.
- **(F)** Discriminative Particle Filter Reinforcement Learning (**DPFRL**) [19] is an on-policy agent that summarizes the history using a differentiable particle filter. DPFRL is one of state-of-the-art model-free POMDP methods.
- **(F)** Variational Recurrent Model (**VRM**) [9] is one of state-of-the-art off-policy model-based agents. It solves POMDPs by using a recurrent variational dynamic model and an SAC agent.
- **(F) Addition baselines** are explored in Appendix G.

6.3 Learning Performance

We compare the success rates and the wall-clock training time of all agents. We alternate training and testing for all agents and compute the average success rates over 100 test episodes for every 2000 environment timesteps.

Success Rates. For HILMO agents, we only report the best performance achieved with a specific strategy to create top-level observations (Full, Final, or

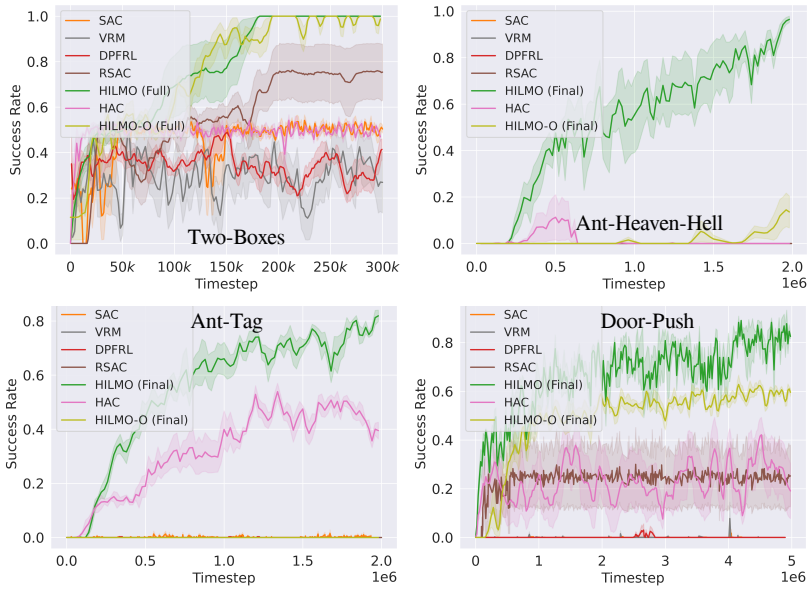


Fig. 5: Success rate means and standard deviations (4 seeds).

Recurrent). From Fig. 5, we can see that across all domains, HILMO consistently outperforms all flat baselines and is on par or better than HILMO-O. Moreover, HILMO is the only agent that can learn well in `Ant-Heaven-Hell` and `Ant-Tag`.

Efficient Exploration. We hypothesize that HILMO performs better because of a more effective exploration. To validate, we visualize the ant's positions in `Ant-Heaven-Hell` during 500k environment timesteps of training in Fig. 6. Apparently, HILMO explores better thanks to high-level actions, covering the T-shaped space densely.

Fig. 6: Coverage of HILMO, HILMO-O, and RSAC in `Ant-Heaven-Hell` after 500k timesteps of training.

HILMO-O also utilizes high-level actions, but its bottom policy rarely achieves given goals, which is detrimental to the final performance. The reason is that without penalizing unachieved goals, the top policy of HILMO-O is free to propose unrealistic goals (*e.g.*, outside of the working space). This issue is also present in the original HIRO agents (see this hyper-link). In `Two-Boxes` and `Door-Push`, the bottom policy of HILMO-O performs better, therefore, its performance is relatively comparable to that of HILMO.

Other Baselines. With no memory, SAC could not solve any domains as expected. Surprisingly, a memory-less HAC can perform quite well in `Ant-Tag` due to a well-trained bottom policy that sometimes can corner and tag the opponent. RSAC can only succeed in `Two-Boxes` for some seeds. It struggles to learn in the

Table 2: Wall-clock training time in *hours* for selected agents in Fig. 5.

Domain	RSAC	HILMO-O	HILMO
`Two-Boxes`	3.31 ± 0.2	$\mathbf{0.96 \pm 0.2}$	1.04 ± 0.3
`Ant-Heaven-Hell`	71.02 ± 0.3	11.31 ± 0.2	$\mathbf{4.69 \pm 0.3}$
`Ant-Tag`	68.25 ± 0.4	12.51 ± 0.3	$\mathbf{6.41 \pm 0.3}$
`Door-Push`	112.7 ± 0.4	26.10 ± 0.2	$\mathbf{23.2 \pm 0.2}$

other tasks in which episodes last longer, and the reward sparsity will hinder learning more severely. DPFRL and VRM surprisingly perform poorly across domains. These methods have no mechanisms to deal with sparse rewards and were tested only on POMDPs that require no active information gatherings (*e.g.*, flickering Atari games, or locomotion tasks with hidden velocities or positions).

Wall-clock Training Time. To measure the time fairly, we train only one experiment at a time on the same CPU (Intel i7-8700K 3.7GHz with 12 processors) and GPU (Nvidia GeForce GTX 1080 8GB). Moreover, we excluded other baselines that did not learn or use GPU in their implementations. Table 2 shows that our agents (HILMO and HILMO-O) take significantly less time to train than RSAC in all domains. While they are relatively comparable in `Two-Boxes`, HILMO-O is slower than HILMO in the remaining three domains. In these domains, while RSAC does not learn after days of training, HILMO can learn good policies in a reasonable time. The acceleration can be attributed to shorter top-level episodes for HILMO because each top-level observation summarizes several primitive observations. Moreover, the bottom policy trained in parallel can learn significantly faster due to full observability (see Appendix H).

6.4 Comparing Full, Final, and Recurrent

The comparison is depicted in Fig. 7. Generally, *Final* (red) is the best strategy, potentially due to more concise information. It dominates in three out of four domains, only be outperformed by *Full* (green) in `Two-Boxes`. *Final* does not perform well in `Two-Boxes` possibly because the proposed goals must be precisely on top of the two boxes for the final observation to contain angular changes. In contrast, other strategies are less restricted. *Recurrent* (purple) introduces another recurrent component into the agent, which seems to hinder learning.

6.5 Robot Experiments for Two-Boxes

The learned policy is deployed on a UR5e robot arm (Fig. 8 left) with a specialized gripper [24] that can control the compliance of its two fingers using hydrostatic actuators. However, we only use the left finger and keep it compliant at all times. We adjust the distance between the fingertip and the table so that deflected angles behave similarly in the simulation. We use wooden boxes of two sizes, making up four configurations, and perform three runs for each

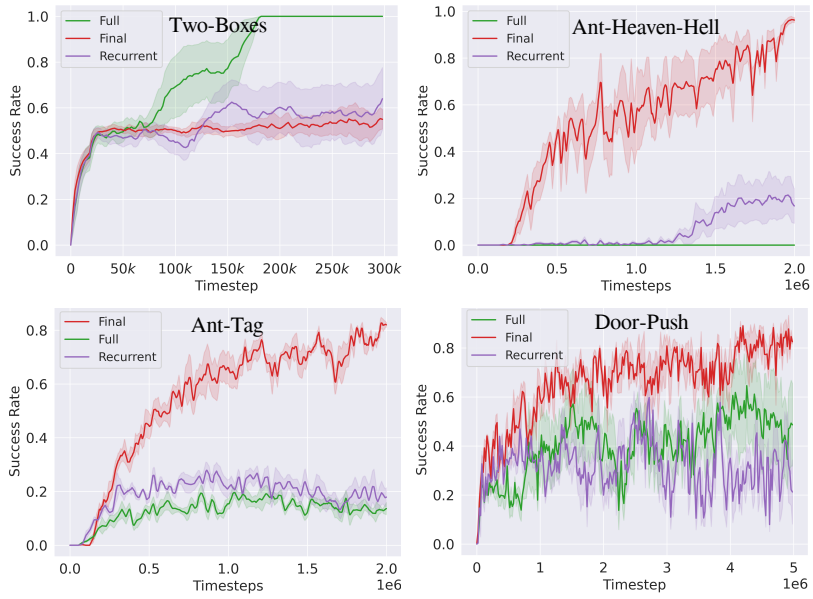


Fig. 7: Comparing the success rates of different top-level observations (4 seeds).

configuration with a random initial position of the finger between the two boxes. All runs are successful. Please refer to https://youtu.be/KiVIBkdm0U8 for the demonstration of the policy as well as the policies learned in other domains.

Policy Visualization. The right side of Fig. 8 illustrates the policy learned by our agent in `Two-Boxes` with two configurations of boxes: small-small and small-big. The learned policy generates both informative and rewarding actions based on the history of the left finger's angles and positions. The agent first goes right until passing a box; then, it backtracks until passing the other box. If the angle history indicates that the two boxes have the same size, the agent will go to the right end through goal 6 in case A to finish the task. Otherwise, it will go to the left end through goal 4 in case B. In `Ant-Heaven-Hell`, we also notice interesting patterns when visualizing the evolution of cell memories of a trained HILMO agent (see Appendix I).

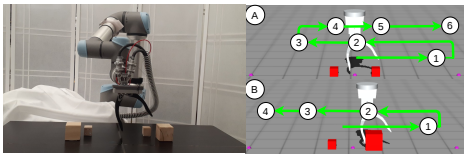


Fig. 8: (Left) A UR5e robot and wooden boxes to deploy the learned policy. (Right) The policy learned in `Two-Boxes` with two distinct box configurations. White circles denote goals generated by the top-level policy.

7 Conclusion and Future Work

This work introduces MOB-MOMDPs, a subclass of MOMDPs which can be found all across active robotic research areas, in which the agent's actions only

have a direct effect on the fully observable component of the state. We also introduce HILMO, a hierarchical agent that exploits the mixed observability assumptions of MOB-MOMDPs. Our empirical evaluation shows that HILMO achieves improved learning performance and training time. A policy learned entirely in simulation is effectively deployed on real hardware.

Even we focus on continuous control tasks, extending our approach for discrete control tasks is straightforward. Moreover, a hierarchical agent further allows state abstractions in which the input might be optimized for each level for more efficient learning. For instance, the top policy can propose optimal goals in `Ant-Tag` and `Ant-Heaven-Hell` without using the joints' angles and velocities.

Acknowledgements This material is supported by the Army Research Office award W911NF20-1-0265 and the NSF grant 1816382.

References

1. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W.: Hindsight experience replay. arXiv preprint arXiv:1707.01495 (2017) 6
2. Astrom, K.J.: Optimal control of markov decision processes with incomplete state estimation. J. Math. Anal. Applic. **10**, 174–205 (1965) 3
3. Chen, M., Nikolaidis, S., Soh, H., Hsu, D., Srinivasa, S.: Planning with trust for human-robot collaboration. In: Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction. pp. 307–315 (2018) 1
4. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014) 7
5. Chung, T.H., Hollinger, G.A., Isler, V.: Search and pursuit-evasion in mobile robotics. Autonomous robots **31**(4), 299–316 (2011) 1
6. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. Journal of artificial intelligence research **13**, 227–303 (2000) 9
7. Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning. pp. 1587–1596. PMLR (2018) 6, 7
8. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. pp. 1861–1870. PMLR (2018) 6, 11
9. Han, D., Doya, K., Tani, J.: Variational recurrent models for solving partially observable control tasks. In: 8th International Conference on Learning Representations, ICLR (2020) 2, 11
10. Heess, N., Hunt, J.J., Lillicrap, T.P., Silver, D.: Memory-based control with recurrent neural networks. arXiv preprint arXiv:1512.04455 (2015) 7
11. Hernandez-Gardiol, N., Mahadevan, S.: Hierarchical memory-based reinforcement learning. Advances in Neural Information Processing Systems pp. 1047–1053 (2001) 9
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997) 7

13. Hsiao, K., Kaelbling, L.P., Lozano-Perez, T.: Grasping pomdps. In: Proceedings 2007 IEEE International Conference on Robotics and Automation. pp. 4685–4692. IEEE (2007) 1
14. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. Advances in neural information processing systems **29**, 3675–3683 (2016) 9
15. Le, T.P., Vien, N.A., Chung, T.: A deep hierarchical reinforcement learning algorithm in partially observable markov decision processes. Ieee Access **6**, 49089–49102 (2018) 9
16. Levy, A., Konidaris, G.D., Jr., R.P., Saenko, K.: Learning multi-level hierarchies with hindsight. In: 7th International Conference on Learning Representations, ICLR (2019) 3, 7, 9, 11
17. Li, A.C., Florensa, C., Clavera, I., Abbeel, P.: Sub-policy adaptation for hierarchical reinforcement learning. In: 8th International Conference on Learning Representations, ICLR (2020) 9
18. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015) 6
19. Ma, X., Karkus, P., Hsu, D., Lee, W.S., Ye, N.: Discriminative particle filter reinforcement learning for complex partial observations. In: 8th International Conference on Learning Representations, ICLR (2020) 2, 11
20. Nachum, O., Gu, S.S., Lee, H., Levine, S.: Data-efficient hierarchical reinforcement learning. In: Advances in Neural Information Processing Systems. vol. 31 (2018) 7, 9, 11
21. Nikolaidis, S., Zhu, Y.X., Hsu, D., Srinivasa, S.: Human-robot mutual adaptation in shared autonomy. In: 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI. pp. 294–302. IEEE (2017) 1
22. Ong, S.C., Png, S.W., Hsu, D., Lee, W.S.: Pomdps for robotic tasks with mixed observability. In: Robotics: Science and Systems. vol. 5, p. 4 (2009) 2, 3
23. Precup, D., Sutton, R.S.: Multi-time models for temporally abstract planning. Advances in neural information processing systems **10** (1997) 6
24. Schwarm, E., Gravesmill, K.M., Whitney, J.P.: A floating-piston hydrostatic linear actuator and remote-direct-drive 2-dof gripper. In: 2019 international conference on robotics and automation (ICRA). pp. 7562–7568. IEEE (2019) 13
25. Steckelmacher, D., Roijers, D.M., Harutyunyan, A., Vrancx, P., Plisnier, H., Nowé, A.: Reinforcement learning in pomdps with memoryless options and option-observation initiation sets. In: Thirty-second AAAI conference on artificial intelligence (2018) 9
26. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012) 9, 10
27. Wang, N., Pynadath, D.V., Hill, S.G.: The impact of pomdp-generated explanations on trust and performance in human-robot teams. In: AAMAS. pp. 997–1005 (2016) 1
28. Wiering, M., Schmidhuber, J.: Hq-learning. Adaptive Behavior **6**(2), 219–246 (1997) 9
29. Xiao, Y., Katt, S., ten Pas, A., Chen, S., Amato, C.: Online planning for target object search in clutter under partial observability. In: 2019 International Conference on Robotics and Automation (ICRA). pp. 8241–8247. IEEE (2019) 1
30. Yang, Z., Nguyen, H.: Recurrent off-policy baselines for memory-based continuous control. Deep RL Workshop NeurIPS (2021) 11

A HIerarchical reinforcement Learning under Mixed Observability (HILMO)

Algorithm 1 HILMO

Constants: Horizons: H (whole agent), H^T (top-level), $H^{\mathcal{X}}$ (bottom-level, *i.e.*, k), goal testing probability: λ
Replay buffers: $\mathcal{D}^T$, $\mathcal{D}^{\mathcal{X}}$; actors and critics $(\pi^T, Q^T), (\pi^{\mathcal{X}}, Q^{\mathcal{X}})$
for as many episodes **do**
    Run-Top(λ)
    Update π^T, Q^T using RDPG [3] from episodes in $\mathcal{D}^T$
    Update $\pi^{\mathcal{X}}, Q^{\mathcal{X}}$ using DDPG [7] from transitions in $\mathcal{D}^{\mathcal{X}}$
end for
function RUN-TOP()
    Empty storage $\tau \leftarrow \emptyset$, history $h^T \leftarrow h^T_{\text{init}}$, top observation $o^T \leftarrow o^T_{\text{init}}$
    for H steps or until the environment solved **do**
        $x^g \leftarrow \pi^T(h^T) +$ exploration noise ▷ Sample a goal for the bottom policy
        test-goal $\leftarrow$ True w.p. λ
        $[o^T, x^g, o'^T, r] =$ Run-Bottom(x^g, test-goal)
        if x^g is tested and missed **then**
            $\tau \leftarrow \tau \cup [o^T, x^g, o'^T, r = -H^T]$ ▷ Store a subgoal testing transition
            $\tau \leftarrow \tau \cup [o^T, x^g_{\text{met}}, o'^T, r]$ ▷ Store a hindsight action transition
        else
            $\tau \leftarrow \tau \cup [o^T, x^g, o'^T, r]$
        end if
        $h^T \leftarrow h^T \cup o'^T$ ▷ Use recurrent module
        $o^T \leftarrow o'^T$
    end for
    Process transitions in τ (Section 4.3) to create episodes and store in $\mathcal{D}^T$
end function
function RUN-BOTTOM(x^g, test-goal)
    $x, z \leftarrow o$; $g \leftarrow x^g$ ▷ Set observation and goal for the bottom level
    $E \leftarrow \emptyset$ ▷ Empty storage for hindsight goal transitions (HGT)
    for $H^{\mathcal{X}}$ attempts or until g achieved **do**
        $a \leftarrow \pi^{\mathcal{X}}(x, g)$ + exploration noise (if not test-goal)
        Execute a in environment, observe $o' = (x', z')$ and environment reward r
        $\mathcal{D}^{\mathcal{X}} \leftarrow [x, a, x', r^{\mathcal{X}} \in \{-1, 0\}, g]$ ▷ Store transitions with intrinsic rewards
        $E \leftarrow [x, a, x', r^{\mathcal{X}} = \emptyset, g = \emptyset]$ ▷ Store HGTs with empty rewards & goals
        $x \leftarrow x'$
    end for
    $\mathcal{D}^{\mathcal{X}} \leftarrow$ Complete HGTs ▷ Determine goals and rewards for HGTs in E
    Create next top-level observation o'^T from all o-s using Full, Final, or Recurrent
    return $[o^T, x^g, o'^T, \sum r]$ ▷ Return a top transition (o^T from Run-Top)
end function

Parameters:

- $\lambda = 0.3$
- $[H, H^{\mathcal{X}}]$: [400, 20] (for `Ant-Tag`, `Ant-Heaven-Hell`, and `Door-Push`), [100, 12] (`Two-Boxes`)

B Deep Deterministic Policy Gradient (DDPG)

Algorithm 2 DDPG algorithm (given a replay buffer of transitions and without target networks)

1: Initialize critic network $Q(s, a \mid \theta^Q)$ and actor $\mu(s \mid \theta^\mu)$ with weights θ^Q and θ^μ
2: Given a replay buffer R
3: **for** M episodes **do**
4: Sample a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
5: Calculate target

$$y_i = r_i + \gamma Q\left(s_{i+1}, \mu(s_{i+1} \mid \theta^\mu) \mid \theta^Q\right)$$

6: Update critic by minimizing the loss

$$\mathcal{L} \approx \frac{1}{N} \sum_i \left(y_i - Q(s_i, a_i \mid \theta^Q)\right)^2$$

7: Update actor using sampled policy gradient:

$$\Delta_{\theta^\mu} J \approx \frac{1}{N} \Delta_a Q(s, a \mid \theta^Q)\big|_{s=s_i, a=\mu(s_i)} \Delta_{\theta^\mu} \mu(s \mid \theta^\mu)|_{s_i}$$

8: **end for**

Implementation details:

- Actor network architecture: (FC-64 + ReLU) + (FC-64 + ReLU) + (FC-action-dim + Tanh)
- Critic network architecture: (FC-64 + ReLU) + (FC-64 + ReLU) + (FC-1)
- Replay buffer: 100k transitions
- Batch size: 1024
- Optimizer: Adam [5] with a learning rate of 0.001 and other default parameters

C Recurrent Deterministic Policy Gradient (RDPG)

Algorithm 3 RDPG algorithm (given a replay buffer of episodes)

1: Initialize critic network $Q^{\omega}(a_t, h_t)$ and actor $\pi^{\theta}(h_t)$ with parameters ω and θ
2: Initialize target networks $Q^{\omega'}$ and actor $\pi^{\theta'}$ with parameters $\omega' \leftarrow \omega$ and $\theta' \leftarrow \theta$
3: Given a replay buffer R of episodes
4: **for** M episodes **do**
5: Sample a minibatch of N episodes from R
6: Construct histories $h_t^i = (o_1^i, a_1^i, \ldots, a_{t-1}^i, o_t^i)$
7: Compute target values for each sample episode $(y_1^i, \ldots, y_T^i)$ using the recurrent target networks

$$y_t^i = r_t^i + \gamma Q^{\omega'}(h_{t+1}^i, \pi^{\theta'}(h_{t+1}^i))$$

8: Update critic (using back-propagation through time)

$$\Delta\omega = \frac{1}{NT} \sum_i \sum_t \left(y_t^i - Q^{\omega}(h_t^i, a_t^i) \right) \frac{\partial Q^{\omega}(h_i^t, a_t^i)}{\partial \theta}$$

9: Update actor

$$\Delta\theta = \frac{1}{NT} \sum_i \sum_t \frac{\partial Q^{\omega}\left(h_t^i, \pi^{\theta}(h_t^i)\right)}{\partial a} \frac{\partial \pi^{\theta}(h_i^t)}{\partial \theta}$$

10: Update actor and critic using Adam [5]
11: Update target networks

$$\omega' \leftarrow \tau\omega + (1-\tau)\omega'$$
$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

12: **end for**

Implementation details:

- Actor network architecture: (LSTM-64) + (FC-64 + ReLU) + (FC-action-dim + Tanh)
- Critic network architecture: (LSTM-64) + (FC-64 + ReLU) + (FC-1)
- Replay buffer: from 5k-10k episodes
- Batch size: 256
- Optimizer: Adam [5] with a learning rate of 3e-4 and other default parameters

D Different Design Choices

GRU v.s. LSTM. We compare the performance of HILMO in `Ant-Tag` and `Two-Boxes` when using GRU [1] and LSTM [4] in Fig. 1. Using GRU results in a slower speed of learning in `Ant-Tag`, but there is no major difference in `Two-Boxes`.

Fig. 1: Performance comparison of HILMO when using LSTM and GRU for the top policy (4 seeds).

RDPG v.s. RTD3 for Top Policy. We compare the performance when using RDPG and RTD3 to learn the top-level policy in `Ant-Tag` and `Two-Boxes` in Fig. 2. RDPG outperformed RTD3 in the two domains, therefore we chose RDPG which is simpler to implement.

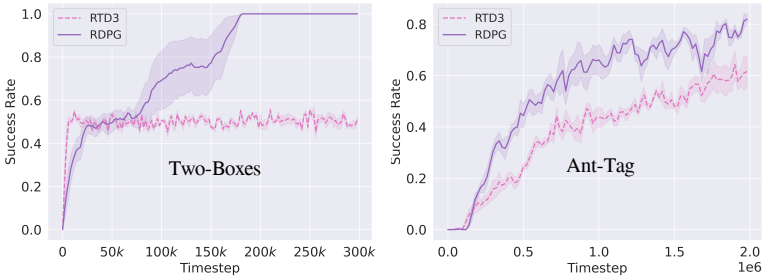
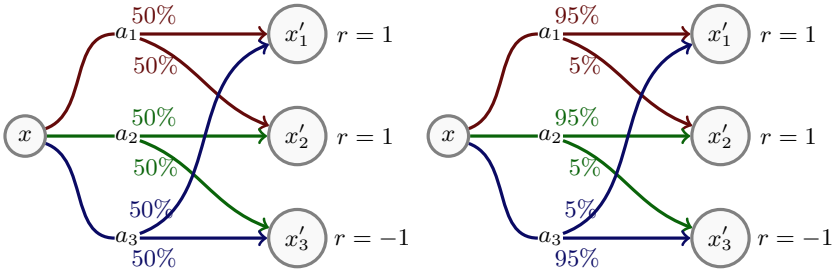


Fig. 2: Using RTD3 and RDPG to learn the top policy (4 seeds).

E Stochastic MOB-MOMDPs and HILMO Optimality

Theorem 1 assumes *deterministic* MOB-MOMDPs, *i.e.*, that $T^{\mathcal{X}}$ is deterministic, while $T^{\mathcal{Y}}$ is free to be stochastic (see Section 3). Here we provide a highly stochastic counterexample that shows why such deterministic transitions are necessary to guarantee the optimality of the HILMO approach. Finally, we will argue that small levels of stochasticity (as might be found in more realistic navigation tasks) do not constitute a significant concern and should still imply that the HILMO approach is quasi-optimal.



(a) High stochasticity. (b) Low stochasticity.

Fig. 3: Examples of stochastic transitions in $\mathcal{X}$-space.

High Stochasticity. Consider the local dynamics depicted in Fig. 3a, and assume that all other rewards are 0, such that the optimality of a policy is exclusively determined by the behavior at x. In such a situation, it is desirable to reach either x'_1 or x'_2 and to avoid x'_3. Given the stochastic dynamics in the example, this means that action a_1 is optimal, while a_2 and a_3 are suboptimal. Note that a_1 does not guarantee to reach any specific x'; it just guarantees that x'_3 is avoided. In the HILMO approach, the closest possible behavior would be to choose either x'_1 or x'_2 as the next "pose". However, if the top-level policy selects that it wants to reach x'_1, the issue arises that x'_1 can be reached by either a_1 or a_3, and neither action is better than the other for the goal of reaching x'_1. Therefore, the low-level policy is unable to prefer a_1 compared to a_3. Similarly, if the top-level policy selects that it wants to reach x'_2, the issue arises that x'_1 can be reached by either a_1 or a_2, and neither action is better than the other for the goal of reaching x'_1. Therefore, the low-level policy is unable to prefer a_1 compared to a_2. Note that the optimal behavior of choosing a_1 to reach x'_1 or x'_2 *can* be represented as a HILMO policy, it just is not intrinsically preferred to some other suboptimal behaviors. In relation to Theorem 1, the issue is that under such stochasticity, the HILMO criteria for the low-level policy considers optimal and suboptimal behaviors to be equally valid.

This issue is potentially resolved by extending the HILMO approach to let the high-level policy select a whole subset of possible goals (or a smooth preference over goals) rather than a single goal, and let the low-level policy select actions which satisfy the aggregate goal as much as possible. Such an extension is left for future work. Instead, we continue the analysis, focusing on a low stochasticity example to show that the HILMO is able to handle low levels of stochasticity.

Low Stochasticity. Consider now the local dynamics depicted in Fig. 3b, *i.e.*, the same scenario described above, except that the transition probabilities associated with the actions have changed to be less stochastic. In this case, action a_1, a_2, and a_3 are respectively more likely (albeit not guaranteed) to cause transitions to x'_1, x'_2, and x'_3. A real-world analogy for such a situation could be a robot moving imperfectly through an environment due to minor wheel slippage. In such a situation, if the top-level policy selects that it wants to reach x'_1, then the low-level policy has a concrete reason to prefer a_1 over a_3, i.e., that a_1 is more likely to transition to x'_1. On the other hand, if the top-level policy selects that it wants to reach x'_2, then the low-level policy has a concrete reason to prefer a_2 over a_1, meaning that there is still a chance of reaching x'_3. However, the top-level policy can take the low-level behavior into account and learn that choosing x'_1 as a target is preferred to x'_2 because choosing x'_1 as target never leads to negative rewards. Therefore, optimal behavior is still guaranteed to be valued better than non-optimal behavior by the HILMO criteria.

We argue that, in practice, realistic motion-based tasks such as the ones modeled by MOB-MOMDPs tend to have low motion stochasticity rather than high motion stochasticity. It follows that the HILMO approach should still be able to achieve quasi-optimal or even optimal behavior, even with low levels of stochasticity.

F HILMO with Off-Policy Corrections (HILMO-O)

Algorithm 4 HILMO-O

Constants: Horizons: H (whole agent), $H^{\mathcal{X}}$ (bottom, *i.e.*, k)
Replay buffers: $\mathcal{D}^T$, $\mathcal{D}^{\mathcal{X}}$; actors and critics $(\pi^T, Q^T), (\pi^{\mathcal{X}}, Q^{\mathcal{X}})$
for as many episodes **do**
    Run-Top()
    Update π^T, Q^T using RDPG from Off-Policy-Correct(episodes in $\mathcal{D}^T$, $\pi^{\mathcal{X}}$)
    Update $\pi^{\mathcal{X}}, Q^{\mathcal{X}}$ using DDPG from transitions in $\mathcal{D}^{\mathcal{X}}$
end for
function RUN-TOP()
    Empty storage $\tau \leftarrow \emptyset$, history $h^T \leftarrow h^T_{\text{init}}$, top observation $o^T \leftarrow o^T_{\text{init}}$
    for H steps or until the environment solved **do**
        $x^g \leftarrow \pi^T(h^T) +$ exploration noise ▷ Sample a goal for the bottom policy
        $[o^T, x^g, o'^T, r], \{x_{1:c}, a_{1:c}, g_{1:c}\} =$ Run-Bottom(x^g)
        $\tau \leftarrow \tau \cup [o^T, x^g, o'^T, r,$ off-policy correction info $= \{x_{1:c}, a_{1:c}, g_{1:c}\}]$
        $h^T \leftarrow h^T \cup o'^T$ ▷ Use recurrent module
        $o^T \leftarrow o'^T$
    end for
    $\mathcal{D}^T \leftarrow \mathcal{D}^T \cup \tau$
end function
function RUN-BOTTOM(x^g)
    $x, z \leftarrow o$; $g \leftarrow x^g$; $c \leftarrow 0$
    for $H^{\mathcal{X}}$ attempts or until g achieved **do**
        $a \leftarrow \pi^{\mathcal{X}}(x, g) +$ exploration noise ▷ Sample a noisy action
        Execute a in environment, observe $o' = (x', z')$ and environment reward r
        $g' \leftarrow x + g - x'$ ▷ Update relative goal
        $\mathcal{D}^{\mathcal{X}} \leftarrow [x, g, a, r^{\mathcal{X}} = -\|g'\|_2, x', g']$ ▷ Store transitions with intrinsic rewards
        $x, g \leftarrow x', g'$
        c += 1
    end for
    Create next top-level observation o'^T from all o-s using Full, Final, or Recurrent
    return $[o^T, x^g, o'^T, \sum r]$, $\{x_{1:c}, a_{1:c}, g_{1:c}\}$
end function
function OFF-POLICY-CORRECT$(\{\tau_i\}, \pi^{\mathcal{X}})$ ▷ For top-level
    for each transition in $\{\tau_i\}$ **do**
        $\tilde{x}^g = \max_{g_1} \prod_{t=1}^{c} \pi^{\mathcal{X}}(a_t \mid x_t, g_t)$ ▷ Find a goal that makes $\pi^{\mathcal{X}}$ take the same actions as its past version
        Replace transition with $[o^T, \tilde{x}^g, o'^T, r]$
    end for
    return $\{\tau_i\}$
end function

Parameters:

- Number of goal candidates used in maximization: 10 (as in HIRO [8])
- Bottom reward scale: 1. Top reward scale: 1 (0.1 was used for HIRO; due to a different reward function)
- $[H, H^{\mathcal{X}}]$: [400, 20] (for `Ant-Tag`, `Ant-Heaven-Hell`, and `Door-Push`), [100, 12] (`Two-Boxes`)

G Additional Baselines

We run additional experiments to measure the success rates (see Fig. 4) of the following additional baselines:

- Recurrent Proximal Policy Optimization (RPPO) is the recurrent version of PPO [9] implemented in [6]
- Recurrent Twin Delayed Deep Deterministic (RTD3) is the recurrent version of TD3 [2] from [10]
- Recurrent Deterministic Policy Gradient (RDPG) [3] implemented in [10]

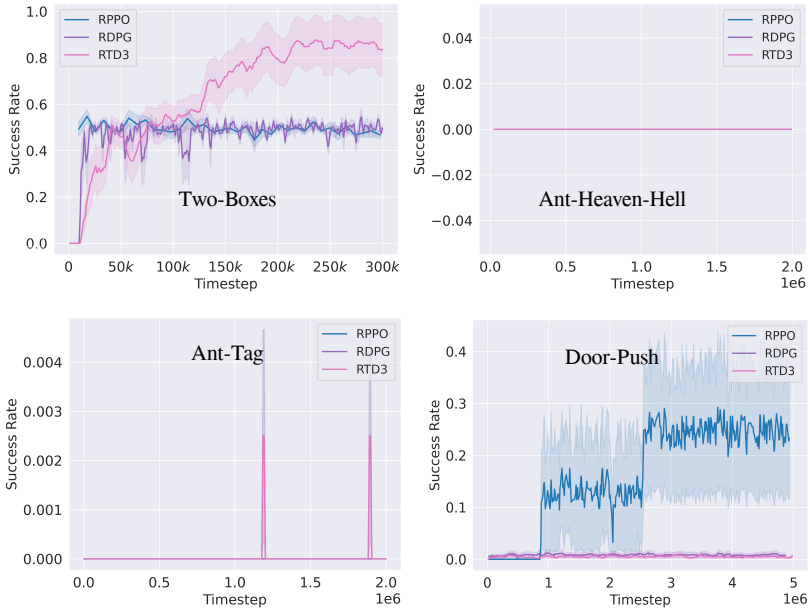


Fig. 4: Success rates of additional baselines (4 seeds).

H Goal Achieved Ratios of Trained Agents

Fig. 5 shows the goal-achieved ratios and the success rates in `Two-Boxes` and `Ant-Heaven-Hell` of trained agents. As expected, the bottom policy with full observability learns quicker than the top policy with the goal-achieved ratio quickly climbing up, indicating the benefit of our approach. More interestingly, the bottom-level policy (red) does not have to be perfect for the whole agent to solve the given tasks satisfactorily. By inspecting the learned policies, we observe that the top-level policy will sometimes propose goals closer to the agent when it realizes that the last goal has not been achieved.

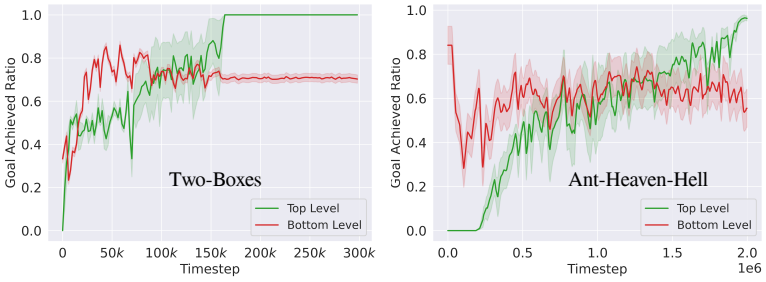


Fig. 5: Goal-achieved ratios at two levels of fully trained agents (4 seeds).

I Evolution of Memory Cells of a Trained Agent

Fig. 6 shows the evolution of several memory cells of the LSTM inside the top-level actor in `Ant-Heaven-Hell` after training. We visualize over four episodes in which heaven is on the left (the left figure) and the right (the right figure). Apparently, memory cells behave differently depending on the side observed when the agent is inside the blue area (marked by shaded areas). The same behaviors generally repeat whenever the same side is observed.

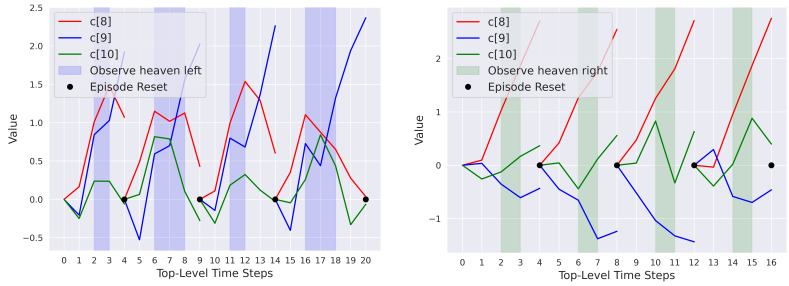


Fig. 6: The memory cell's internal states $c[j]$ of the LSTM network of a trained top-level actor in `Ant-Heaven-Hell` during four episodes with heaven on the left (left figure) and on the right (right figure). The shaded areas mark when the agent is inside the blue area and can observe the side of heaven.

Appendix References

1. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014) 4
2. Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning. pp. 1587–1596. PMLR (2018) 8
3. Heess, N., Hunt, J.J., Lillicrap, T.P., Silver, D.: Memory-based control with recurrent neural networks. arXiv preprint arXiv:1512.04455 (2015) 1, 8
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997) 4
5. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014) 2, 3
6. Kostrikov, I.: Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail (2018) 8
7. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015) 1
8. Nachum, O., Gu, S.S., Lee, H., Levine, S.: Data-efficient hierarchical reinforcement learning. In: Advances in Neural Information Processing Systems. vol. 31 (2018) 7
9. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017) 8
10. Yang, Z., Nguyen, H.: Recurrent off-policy baselines for memory-based continuous control. arXiv preprint arXiv:2110.12628 (2021) 8